

SDEV1201 Database Fundamentals

LESSON 3

Let's review

What is a database?

Databases are an organized collection of data. There are many kinds of databases.

In SDEV1201 we will be using a type of database called a relational database. A relational database is where like information is stored in different tables that all relate to each other in some way

Databases are necessary for large organizations for the following reasons:

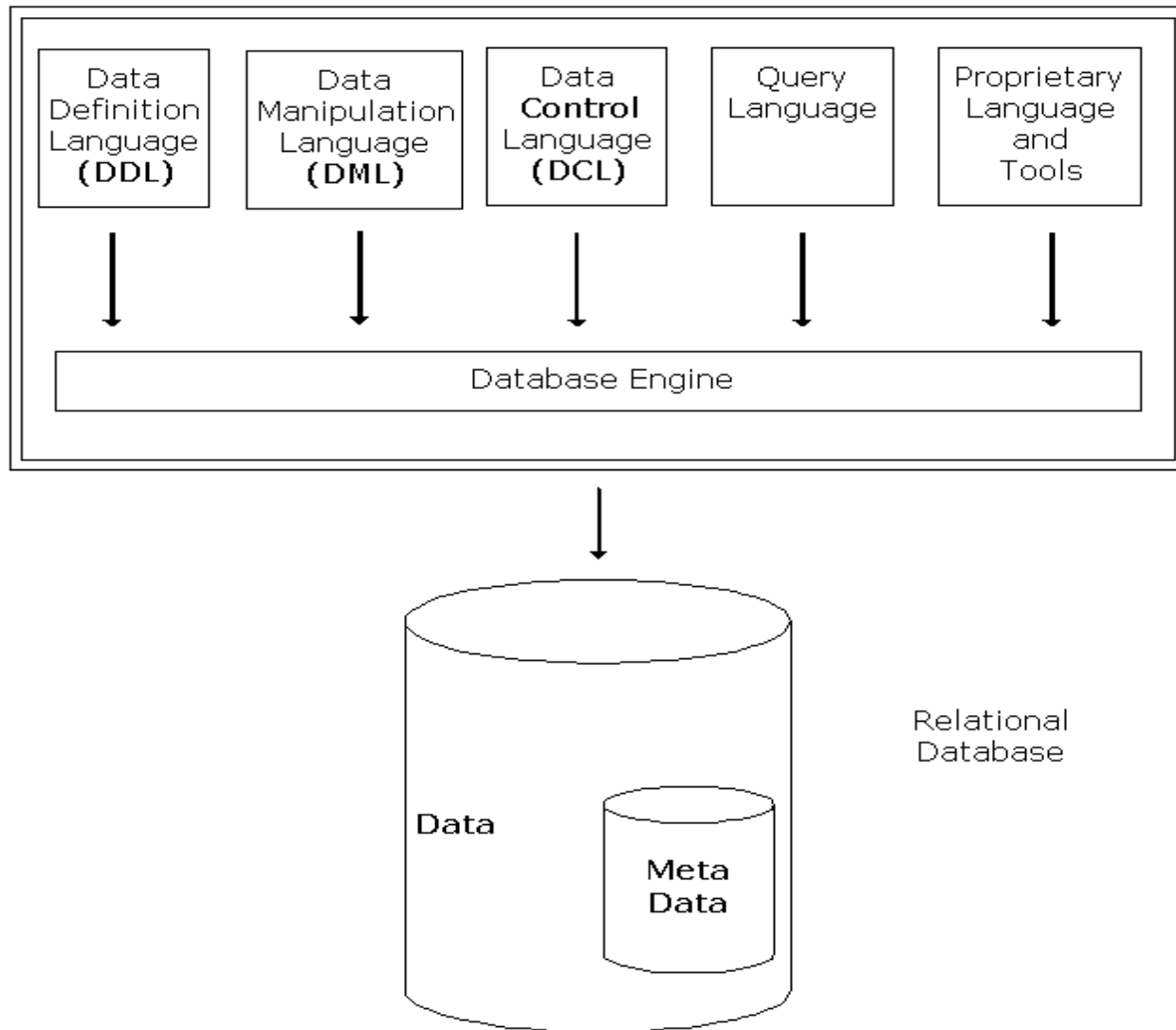
- Size
- Ease of updating
- Accuracy
- Security
- Redundancy
- Importance

Databases and DBMS

Databases are what store the data. To effectively maintain the data we use a Database Management System (DBMS). You would use the DBMS to manage your databases. The database is the data and the rules. The DBMS allows you to interact with your db.



Database Management System (DBMS)



The database engine is the core set of programs that make up a DBMS. These programs execute when a request is submitted to the DBMS and directly interact with the database itself to provide a service. For example, a user adds new data to the database. The user would execute an application program to perform this task. The application program would submit a DML statement to the DBMS requesting that new data be added to the database. The DML statement causes one or more programs in the database engine to execute and actually add the new data to the database. It is important to note that developers and users do not interact directly with the database; instead they submit requests to the DBMS. The DBMS responds to requests from the various stakeholders in the system (e.g. users or developer) and performs the various tasks involved in creating, changing, and using a database.

The Relational DBMS

A relational DBMS uses tables to store data in the database. A table consists of rows and columns (similar to a spreadsheet). For example, information about customers would be stored in a table named Customer in the database as shown below:

<u>CustomerNumber</u>	FirstName	<u>LastName</u>	Address	Phone
100	Adams	Julie	9825-77 Ave Edmonton AB T5C 8E2	780-111-1111
200	Smith	Miles	4308-133 Ave Edmonton AB T5C 8E9	780-222-2222

Entity Relationship Model

The **Entity-Relationship** model is a tool that organizes and documents the logical design of a database. The entity-relationship model describes the data used by an application in terms of: **entities, attributes, and relationships.**

Entity Relationship Model

Entity

An entity is a person, place, thing or concept that is used in the business context of the application and for which data is collected.

Attributes

A piece of data that describes an entity is called an **attribute**. Synonyms for attribute include: element, property and field. In the order example, the customer entity has the following attributes:

- Customer Number
- Last Name
- First Name
- Address
- Phone Number

The values for all attributes describing an entity make up one instance of the entity. For example, the following values describe one specific customer.

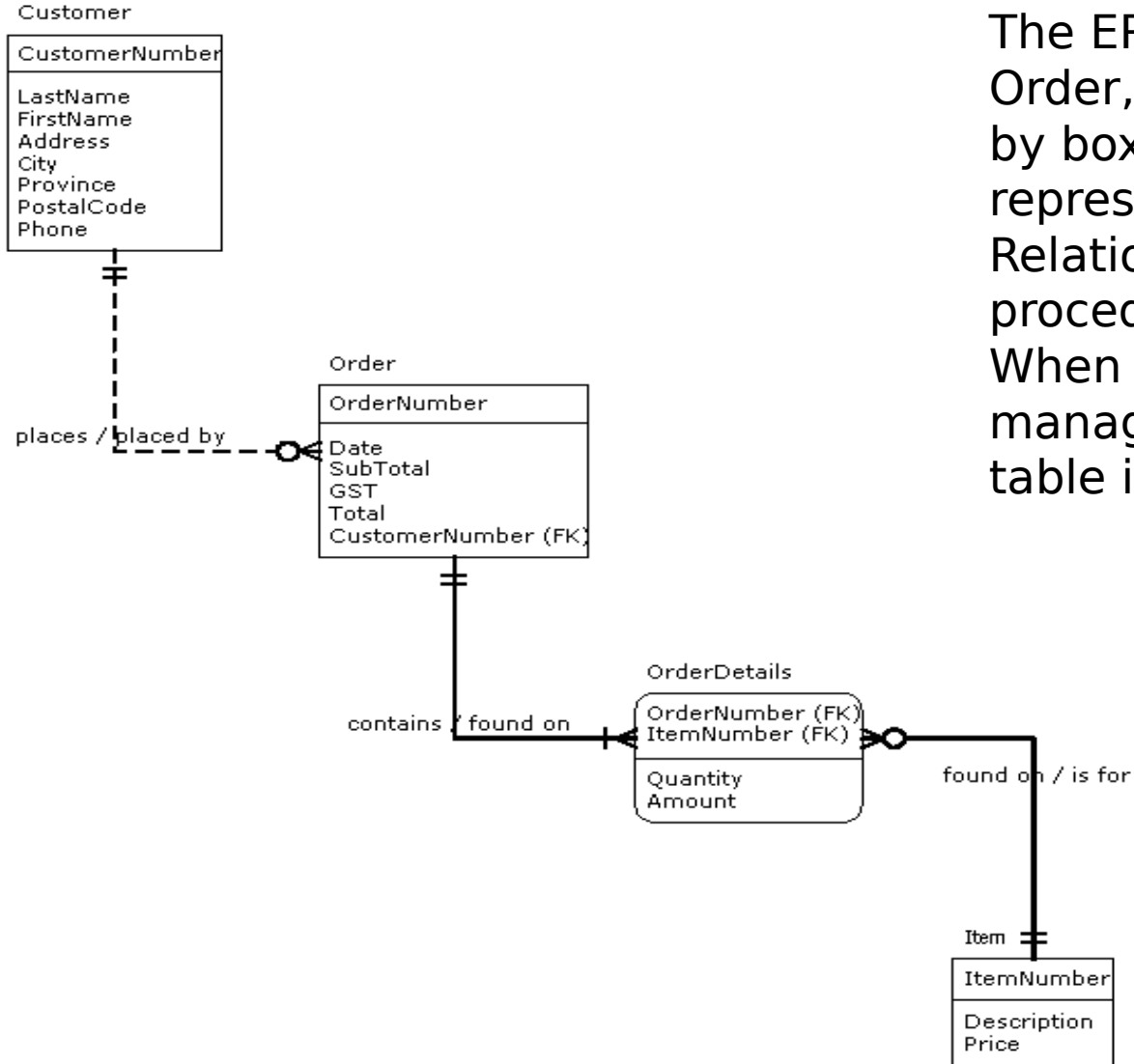
Customer Number	Last Name	First Name	Address	Phone
100	Adams	Joan	9825-77 Ave Edm Ab T5C 8E2	488-8712

Entity Relationship Model

The logical database design process uses the entity-relationship model to represent the business scenario as a collection of entities and relationships. The entity-relationship diagram (ERD) documents the logical design and becomes a starting point for the physical design process, which defines the structure of the database.

The **Entity-Relationship** model is a tool that organizes and documents the logical design of a database. The entity-relationship model describes the data used by an application in terms of: **entities, attributes, and relationships.**

Sample ERD



The ERD shows four entities: Customer, Order, OrderDetails and Item, all represented by boxes. The lines connecting the entities represent relationships between the entities. Relationships reflect the business rules and procedures in place within an organization. When using a relational database management system each entity becomes a table in the database.

Values an Attribute Can Assume

Every attribute has a set of legitimate values, defined in terms of the **data type** and **domain** of the attribute.

*e.g. the customer number must be a **number** **greater than zero***

Special Cases: The **NULL** value. **NULL** either means:

1. The value is unknown.
2. The attribute does not apply for this particular instance.

Primary Keys

Primary keys are a **unique** identifier for each instance of an entity. e.g. the Item Number uniquely identifies each item in inventory, and can act as the Primary Key for the Item entity.

Item

ItemNumber	Description	Price
333	Hammer	12.00
777	Saw	15
888	Power Drill	90

A combination of two or more attributes acting as the primary key is called a **concatenated key**.

OrderDetails

OrderNumber	ItemNumber	Quantity	Price	Amount
1	333	3	15	45
2	777	7	10	70
3	888	8	5	40
3	777	3	10	30

Relationships

A relationship represents a business association between two entities. Relationships are based on the business rules of the organization. All relationships are bi-directional in that they can be interpreted from the point of view of each entity. In the customer order example a relationship exists between the customer entity and the order entity. This reflects the fact that each time a customer makes a purchase an order is created. From the customer perspective, one customer can place many orders. From the order perspective, one and only one customer places an order.

Types of Relationships

- One-to-one

- These are very rare.
- e.g. each country has *one UN representative*, and each UN representative represents only *one country*. The Country entity and the UNRepresentative entity have a 1:1 relationship.

- One-to-many

- We'll see these LOTS.
- e.g. each Department has *many employees*, but each employee belongs to a *single department*. The Employee entity and the Department entity have a 1:many relationship.

- Many-to-many

- We'll see these LOTS, and we'll do something special in our design to represent them.
- e.g. a student enrolls in *many classes*, and each class has *many students*. The Student entity and the Class entity have a many:many relationship.

Symbols for Cardinality

Here are the symbols associated with the crow's foot notation:

Zero



The symbol/diagram above denotes zero in crow's foot notation. We know this because of the zero/circle indicator at the right side of the horizontal line.

One



The diagram above shows a horizontal line with a short vertical line crossing it. The vertical line acts as the indicator – it denotes one.

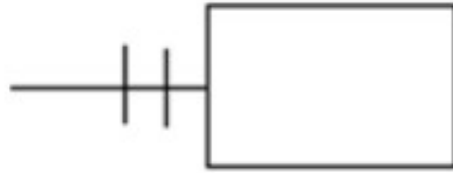
Many



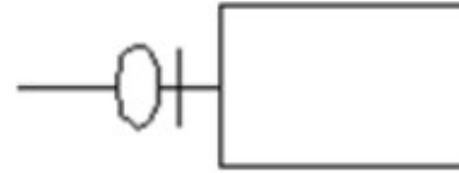
The diagram above denotes many. You can easily remember this symbol because it looks like a crow's foot.

The three diagrams are the basic representation of indicators in crow's foot notation. But in most cases, these indicators are combined to fully understand the relationship between entities.

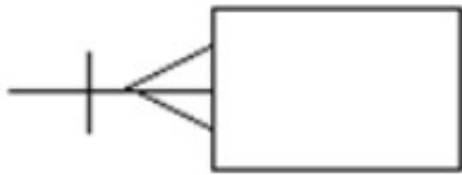
Symbols for Cardinality



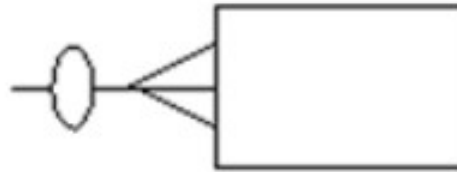
... to one
(mandatory 1)



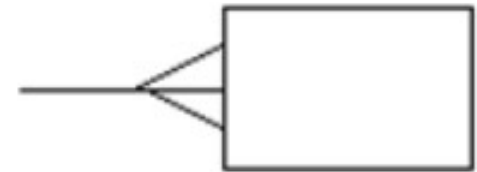
... to zero or one
(optional 1)



... to one or many
(mandatory many)



... to zero, one, or many
(optional many)



... to many

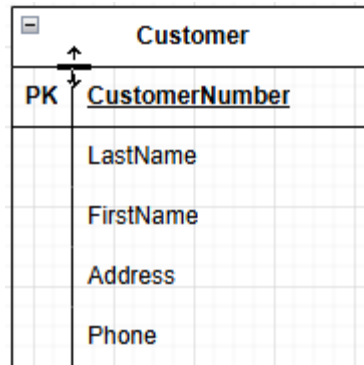
Foreign keys

A **foreign key** is how we define relationships between entities.

A foreign key is a **primary key of one entity** (the “parent”) that appears as an **attribute of another entity** (the “child”).

Note: the two attributes may or may not have the same name. e.g. the Student ID in the Marks table may be called just ID in the Student table.

Entities and Tables



After the ERD design is complete, when the database is created the entities become tables.

<u>CustomerNumber</u>	FirstName	<u>LastName</u>	Address	Phone
100	Adams	Julie	9825-77 Ave Edmonton AB T5C 8E2	780-111-1111
200	Smith	Miles	4308-133 Ave Edmonton AB T5C 8E9	780-222-2222

Terminology

- ❑ Database

An organized collection of data

- ❑ DBMS (Database Management System)

Software system used to maintain the data and manage the database.

- ❑ Business rules (“requirements”)

How the tables relate to each other

- ❑ Entities

Represents a real world thing such as a customer or an order.

- ❑ Attributes

Characteristics of an entity.

Terminology Continued

❑ Relationships

A relationship represents a dependency between entities based on the business rules of the organization.

❑ Entity-relationship diagram

Describes the business scenario in terms of entities and relationships.

❑ Entities

❑ Associative entity

An entity used to accommodate many to many entities.

Terminology continued

- ❑ Attributes
 - ❑ Atomic and composite attributes
 - ❑ Primary key and foreign key
 - ❑ Technical key
- ❑ Parent & child relationship
- ❑ Normalization

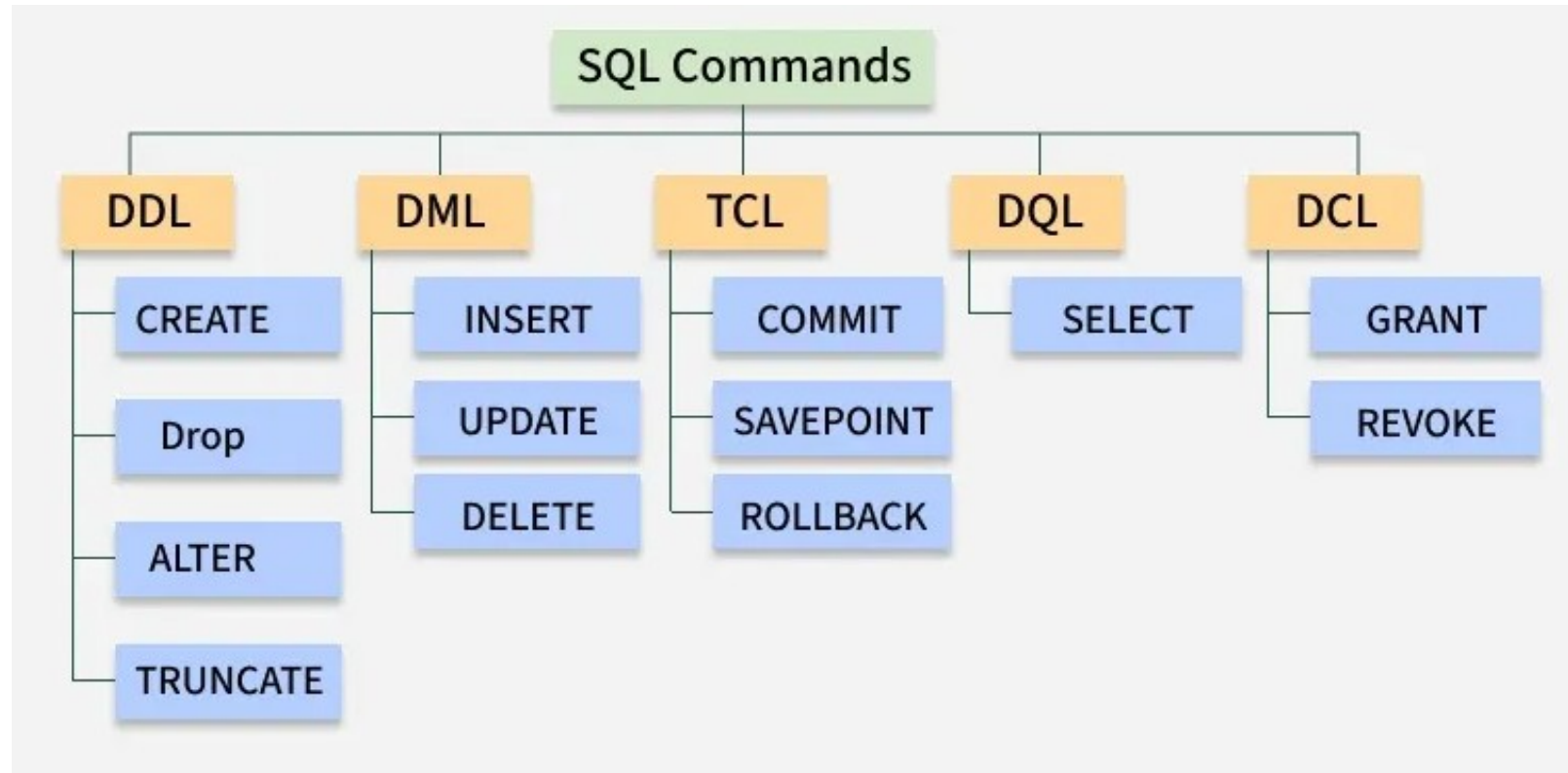
Method to guide us on determining the number of tables and assignment of data items. Minimizes the occurrence of update, insert, and delete anomalies.

Today's Objective

By the end of this section you will be able to understand:

- ❑ Discuss SQL command subsets (DDL/DML).
- ❑ Discuss SQL Style Guidelines.

SQL Commands



SQL Commands – Data Definition Language (DDL)

DDL (Data Definition Language) actually consists of SQL commands that can be used for defining, altering and deleting database structures such as tables, indexes and schemas. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Common DDL Commands

Command	Description	Syntax
<u>CREATE</u>	Create database or its objects (table, index, function, views, store procedure and triggers)	CREATE TABLE table_name (column1 data_type, column2 data_type, ...);
<u>DROP</u>	Delete objects from the database	DROP TABLE table_name;
<u>ALTER</u>	Alter the structure of the database	ALTER TABLE table_name ADD COLUMN column_name data_type;

SQL Commands – Transaction Control Language

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, transaction fails.

Common TCL Commands

Command	Description	Syntax
BEGIN TRANSACTION	 Starts a new transaction	BEGIN TRANSACTION [transaction_name];
COMMIT	Saves all changes made during the transaction	COMMIT;
ROLLBACK	Undoes all changes made during the transaction	ROLLBACK;
SAVEPOINT	Creates a savepoint within the current transaction	SAVEPOINT savepoint_name;

SQL Commands – Data Manipulation Language (DML)

DML commands are used to manipulate the data stored in database tables. With DML, you can insert new records, update existing ones, delete unwanted data or retrieve information.

Common DML Commands

Command	Description	Syntax
<u>INSERT</u>	Insert data into a table	INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
<u>UPDATE</u>	Update existing data within a table	UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
<u>DELETE</u>	Delete records from a database table	DELETE FROM table_name WHERE condition;

SQL Commands – Query Language

DQL is used to fetch data from the database. The main command is SELECT, which retrieves records based on the query. The output is returned as a result set (a temporary table) that can be viewed or used in applications.

DQL Command

Command	Description	Syntax
<u>SELECT</u>	It is used to retrieve data from the database	SELECT column1, column2, ...FROM table_name WHERE condition;
FROM	Indicates the table(s) from which to retrieve data.	SELECT column1 FROM table_name;
<u>WHERE</u>	Filters rows before any grouping or aggregation	SELECT column1 FROM table_name WHERE condition;

SQL Commands – Query Language cont'd

DQL Command

Command	Description	Syntax
<u>GROUP BY</u>	Groups rows that have the same values in specified columns.	SELECT column1, AVG_FUNCTION(column2) FROM table_name GROUP BY column1;
<u>HAVING</u>	 Filters the results of GROUP BY	SELECT column1, AVG_FUNCTION(column2) FROM table_name GROUP BY column1 HAVING condition;
<u>DISTINCT</u>	Removes duplicate rows from the result set	SELECT DISTINCT column1, column2, ... FROM table_name;
<u>ORDER BY</u>	Sorts the result set by one or more columns	SELECT column1 FROM table_name ORDER BY column1 [ASC DESC];
<u>LIMIT</u>	By default, it sorts in ascending order unless specified as DESC	SELECT * FROM table_name LIMIT number;

Basic Select Statement - Queries

Class Preparation

Examples and practice in this class are based on the IQSchool database.

The script to create the tables is located in the Query section of the Brightspace site and named IQSchool Create Tables and Load data script.

The script creates all the tables and constraint and loads data.

Let's create a new IQSchool database, open the script and run it now.

Queries

Queries are used to retrieve data from the database. They can be very flexible in what columns they select, what rows they select, and what aggregate calculations they return. Being able to write accurate and efficient queries is a critical skill when working with databases.

Query Syntax

Queries are written with the SELECT statement. The simplified syntax for a select statement is:

```
SELECT [ALL | DISTINCT] column_list
[FROM table_name [, table_name2 [... , table_name]] -- source of data
      [INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN] -- other sources
[WHERE clause] -- filter
[GROUP BY clause] -- aggregate by
[HAVING clause] -- filter by aggregate
[ORDER BY clause] -- sorts results
```

If it looks complex don't worry. All the "clauses" listed don't need to be used when creating a query. However, what clauses you do use MUST be in the order shown above (i.e. HAVING MUST be after GROUP BY for example and not vice versa).

Query Syntax

In addition, there are other functionalities beyond the syntax such as the ability to use string, date, and aggregate functions in your queries to retrieve and calculate data to be returned. We will start by looking at simple queries and then adding additional clauses after that.

Simple Query

A simple query can select data from one table. The query can select many rows, calculated data, or even hard coded data when needed. Calculated data be mathematical calculations (subtotal * 1.05) or can be calculated string data (firstname || ' ' || lastname). It can even be a combination of them both ('total: ' + subtotal * 1.05). While formatting what you select in your query is not a common practice (e.g. formatting is best left to the application requesting the data) it can be useful sometimes.

Example:

```
Select FirstName, LastName, City  
From Student;
```

As you can see this will select the student's FirstName, LastName and city for all the student records. The From clause indicates the table the columns are in that you are selecting. Queries like this are very simple to create and are quite common.

Simple Query

To select the student's name as one column instead of separate first name and last name columns we can concatenate them together as follows.

```
Select FirstName || ' ' || LastName, City  
From Student;
```

In PostgreSQL, the || operator will concatenate string data together as well as numeric data. Just remember to add a space between the names otherwise you will get only one long name with FirstName and LastName combined. You can also use the CONCAT function, but each value must be a string datatype.

Derived fields need a name!

There is another difference with the query that returns the names separately than the one that returned them as one column. The column in the second query does not have a name. This is a problem. We should give all columns a name. Calculated columns do not have a name because they are calculated from many different columns and values. So, we must give the column a name. In the example, the “AS” keyword is optional.

```
Select FirstName || ' ' || LastName AS "Student Name", City  
From Student;
```

In this course you are mandated to ALWAYS have column names.

A (dangerous) shortcut

Since we are selecting all the columns in the student table, we could use the '*' operator. The asterisk represents all columns when used in the column list of a query. The same query could look like this:

```
Select *  
From Student;
```

While this is useful to quickly check the contents of a table, or as a starting point to develop a more complex query, we will NEVER include this in a final query: always explicitly list the columns you are selecting.

Break

Where

So far, we have only selected data that includes ALL the records on the table. While this is not uncommon, it is likely that you will not only want to select the columns you want, but also only the records you want. The WHERE clause allows you to limit the records that are returned based on criteria you provide.

```
Select StudentID, FirstName, LastName, Gender, StreetAddress, City,  
Province, PostalCode, Birthdate, BalanceOwing  
From Student  
Where StudentID = '198933540';
```

This query would return all the student information for the student with StudentID 198933540. The “Where” clause can contain many different expressions to identify the records the query should return.

Search Criteria Operators and Conditions

Sometimes the conditions that must be met in the data for certain records to be returned involve many expressions that may be combined using AND/OR operators or several other ones.

Operators

The equals sign = is used when looking for an exact match

```
Select BalanceOwing  
From Student  
Where StudentID = ' 199899200';
```

The not equal sign <> or != is used when looking for something that does not match your criteria

```
Select FirstName || ' ' || LastName "Student Name", BalanceOwing  
From Student  
Where BalanceOwing != 0;
```

Other common operators include <, <=, >, >=

Operators

AND: Both expressions must be true for the record to be returned

OR: If either of the expressions are true then the record is returned

```
Select FirstName || ' ' || LastName "Student Name", BalanceOwing  
From Student  
Where City = 'Edmonton' and  
Province = 'AB';
```

```
Select FirstName || ' ' || LastName "Student Name", BalanceOwing  
From Student  
Where City = 'Edmonton' or  
Province = 'SK';
```

Operators

BETWEEN Returns all records where the search criteria is between two values (inclusive). You can also use NOT BETWEEN to return records where the search criteria is outside the range.

```
Select FirstName || ' ' || LastName "Student Name", BalanceOwing  
From Student  
Where StudentID between 198933540 and 199966250;
```

Operators

IN allows you to list a number of valid search values for a column. You can always code the same query with a compound expression with multiple OR conditions but that can get quite large with many values. You may also use NOT to return everything that is NOT in the list.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where StudentID in  
(198933540, 199912010, 200688700, 200978400);
```

Like

The Like operator is used when you use wildcards in your search or perform pattern matching. Wildcards are characters that when used with the Like operator allow the wildcards to represent a character or multiple characters. The Like operator is case sensitive in PostgreSQL.

For example, if we wanted to return all the student's names whose names started with the letter 'D' we would use the % wildcard character. The % wildcard means any character and any number of characters.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName Like 'D%';
```

It is important to use the like operator when using wildcards. The % would be interpreted as a literal '%' if we used = instead of Like.

Like

The `_` (underscore) is a single character wildcard. This would be used when you are looking for any one character in your search criteria. If we wanted the students' names whose first name was 3 characters long and started with 'Ja' we could use this query.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName Like 'Ja%';
```

If we had used a `%` instead of `_` we would also return students whose First Name was longer than 3 characters.

Like

They can also be used in combination with each other. Let's return all the Student Names that start with 'D', the third character is 'n' and end in 's'.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName Like 'D_n%s';
```


Regular Expressions

You can also perform a search on a range of values using the caret symbol (~). A query to return all the student names that start with A through J could look like this. Each set of square brackets is for one character.

The [A-J] means that the first character must be in that range and can be followed by any number of characters (including zero) after that.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName ~ '[A-J]';
```

Regular Expressions

You can add another set of brackets to check if the second character is an "a".

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName ~ '[A-J][a]';
```

Regular Expressions

If you want to check if there are any occurrences of the second character, you add a “.” between the two sets of square brackets.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName ~ '[A-J].*[y]';
```

Regular Expressions

You can also check for a particular position. Let's select all the student names who have a Postal Code that starts with T through Z and ends with zero to three. The carat (^) looks at the beginning of the string and the dollar sign (\$) looks at the end of the string.

```
Select FirstName || ' ' || LastName "Student Name", PostalCode  
From Student  
Where PostalCode ~ '^[T-Z].[0-3]$';
```

Order By

The ORDER BY clause is used to return the results in a certain order. The results can be sorted by one column or more than one column in either descending (DESC) or ascending (ASC) order. When no order is specified, ASC is the default.

There are numerous ways to use ORDER BY.

```
Select FirstName "First Name", LastName "Last Name"  
From Student  
Order By LastName Desc;
```

Returns the Student Names in Descending order by last name (reverse alphabetical).

Order By

```
Select FirstName "First Name", LastName "Last Name"  
From Student  
Order By "Last Name" Desc;
```

Returns the Student Names in Descending order by last name using the alias (i.e. Last Name) we gave to the LastName column.

Order By

```
Select FirstName "First Name", LastName "Last Name"  
From Student  
Order By "First Name" Asc, "Last Name" Desc;
```

Returns the Student Names in Ascending order of First Name and when duplicate first names exist, it sorts those by Descending order of Last Name.

ASC is the default and is commonly accepted so it is not uncommon to see the Order By clause used for Ascending ordering without ASC. However, you must use DESC if you want the results sorted in Descending order.

ALL and DISTINCT

ALL and DISTINCT can be in a query or with any aggregate function. **ALL** is the default and usually not explicitly declared.

```
Select All FirstName, Province  
From Student;
```

Returns all the Student First Names and their Province. This returns 16 records because there are 16 students.

ALL and DISTINCT

Select Distinct FirstName, Province
From Student;

Returns all the unique (Distinct) student First Names and their Province. This returns only 15 records because there are 2 students with a First Name of Joe from Alberta.

When Distinct is used with an aggregate function (i.e. Count) it would count only the unique (i.e. distinct) values.

Class Activity

Do the “Basic Select Exercise 1” found in Brightspace in the “Week 4 – SELECT statements”.

Homework

Work on “Basic Select Exercise 1” found in Brightspace in the “Week 4 – SELECT statements”.

Reminder: “Entity Relationship Diagram Symbols & Terminology” **Quiz** next week **on Monday, September 15, 2025.**